

# The DoD SoftwareTech

WWW.SOFTWARETECHNEWS.COM

## NEWS



AUGUST 2002 Vol.5, No.3

# Experience Based Management

<http://iac.dtic.mil/dacs/>

Unclassified and Unlimited Distribution

# About the DoD Software Tech News

## STN Editorial Board

**Lon R. Dean**  
**Editor**

ITT Industries, DACS

**Paul Engelhart**  
**DACS COTR**

Air Force Research Lab (IFED)

**Elaine Fedchak**  
ITT Industries

**Morton A. Hirschberg**  
**Editorial Board Chairman**  
US Army Research Lab  
(retired)

**Philip King**  
ITT Industries, DACS

**Thomas McGibbon,**  
**DACS Director**  
ITT Industries, DACS

**Dave Nicholls,**  
**DACS Deputy Director**  
ITT Industries, DACS

**Marshall Potter**  
Federal Aviation Administration

## Article Reproduction

Images and information presented in these articles may be reproduced as long as the following message is noted:

“This article was originally printed in the *DoD Software Tech News*, Vol. 5, No. 3. Requests for copies of the referenced newsletter may be submitted to the following address:

Lon R. Dean, Editor  
Data & Analysis Center for Software  
P.O. Box 1400  
Rome, NY 13442-1400

Phone: 800-214-7921  
Fax: 315-334-4964  
E-mail: [news-editor@dacs.dtic.mil](mailto:news-editor@dacs.dtic.mil)

An archive of past newsletters is available at [www.dacs.dtic.mil/awareness/newsletters/](http://www.dacs.dtic.mil/awareness/newsletters/).”

In addition to this print message, we ask that you send us three copies of any document that references any article appearing in the *DoD Software Tech News*.

## About This Publication:

The *DoD Software Tech News* is published quarterly by the Data & Analysis Center for Software (DACS). The DACS is a DoD sponsored Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC) under the Defense Information Systems Agency (DISA). The DACS is technically managed by Air Force Research Laboratory, Rome, NY and operated by ITT Industries, Advanced Engineering and Sciences Division.



**DACS**

P.O. Box 1400  
Rome, NY 13442-1400

Phone: 800-214-7921  
Fax: 315-334-4964  
E-mail: [dacs@dtic.mil](mailto:dacs@dtic.mil)  
URL: <http://iac.dtic.mil/dacs>

Cover Design by  
Steve Lisi, Steffen Publishing



## To Subscribe to this Publication Contact:

Phone: 800-214-7921  
Fax: 315-334-4964  
E-mail: [news-editor@dacs.dtic.mil](mailto:news-editor@dacs.dtic.mil)  
Web: [www.dacs.dtic.mil/forms/regform.shtml](http://www.dacs.dtic.mil/forms/regform.shtml)

## Distribution Statement:

unclassified and unlimited

STN 5-3: Experience Based Management

# Tech Views: The Role of Experience

by Morton Hirschberg, STN Editorial Board Chairman

Funk and Wagnall's defines 'EXPERIENCE' as knowledge derived from ones own action, practice, perception, enjoyment or suffering; how superbly applicable to Software Engineering. The articles in this issue deal with perceptions gained through the actions and communications of the lessons learned from others as well as practice gained individually and in teams. Raw talent can take one far however, when tempered with wisdom, success is more likely. It is the experience of programming teams and their management that leads to success.

Vic Basili (UMD) and Barry Boehm (USC) have been instrumental in the design and implementation of "The Experience Factory." The Experience Factory defines a framework to plan the use of resources, manage and manipulate resources and processes and strives for continual improvement. Key to Experience is its capture, structure, searchability, availability and maintainability. The article by Basili and Boehm details a large program called CeBase and its immediate application which is the Lessons-Learned Repository for COTS software development.

In an article summarizing his keynote address at the 2002 Software Technology Conference (STC) in Salt Lake City, Lloyd K. Mosemann II (SAIC) gives us his perspective on the ills of Software Engineering, especially, 'best practices' and suggestions for healing them. Seldom have I been in such near congruent accord with an author. Bravo!!

John Salasin (DARPA) and his coauthors, Assad Moini (SPC) and Gwen Williams (Schafer Corp.) present Habitats, an Architecture for the Global Information Grid. "The Habitats concept and its supporting technology will enable a new generation of systems that can successfully and predictably operate in a network-centric world of vastly distributed and dispersed resources, devices and users interacting via the Global Information Grid."

In a previous article, Lawrence Bernstein (Stevens Institute of Technology) and his colleagues have introduced a software-engineering course whereby computer science students are exposed to real world problems. His students are given problems which failed in real life, and are expected to work through to a successful solution. A team approach is used. The value of management is clearly brought to bear. Now Mr. Bernstein exposes us to Software Rejuvenation, a fault tolerant process that has had much success.

In searching for experience based and apropos software engineering material, the following web sites contained a great deal of information. First, from the Software Program Managers Network - <http://www.spmn.com/lessons.html>. Second, from Software Methods and Tools - <http://www.methods-tools.com>. Finally, from Brad Appleton - <http://www.bradapp.net>. Especially useful are Brad's many Software Engineering links.



# Lessons-Learned Repository for COTS-Based SW Development

by Victor Basili, Mikael Lindvall, Iona Russ, and Carolyn Seaman,  
Fraunhofer Center for Experimental Software Engineering, Maryland and  
Barry Boehm, University of Southern California

## Abstract

Individuals acquire knowledge while working in software development projects but too often this knowledge is not documented or captured so that it can be shared or later reused. Dissemination of experience related to new development paradigms, such as development based on commercial off-the-shelf (COTS) components, can be particularly useful. To support sharing experiences about COTS-based software (CBS) development, we have built and made available to the software engineering community a web-based repository of lessons learned. This collection of practical knowledge is currently seeded with lessons extracted from published experiences and online Workshops. The repository offers capabilities for online search and retrieval, submission of lessons and feedback, plus a visual query interface that allows for analysis of the lessons learned and their usage. The repository will grow organically in the directions indicated by usage and contributions from its users.

## 1. Introduction

COTS-based software development is different in many respects from in-house software development. Managers and developers must perform new activities and need new skills and knowledge whose acquisition is time consuming. Let us illustrate what we mean with a fictional story.

*Joe Cotsford is a software team lead working for a government contractor and he has a problem: Joe has been charged by his project manager with the task of evaluating and selecting a COTS product to use as a major subcomponent of their CCTwo system. He's solicited all his colleagues on products to evaluate, and received about 20 suggestions. He spent a lot of time collecting information on these 20 products and was able to eliminate about half of them based on functionality, platform, and price. But now he needs some more criteria to use to whittle down the current list of 10 products to just one. He is at a loss as to how to proceed because he can't think of any other relevant factors to use to continue his evaluation. So he looks on the web for any available resources, but is quickly overwhelmed with the number of irrelevant hits. Surely someone somewhere has dealt with this problem before, but where can he find this experience?*

Although this is a hypothetical character and scenario, the situation is not unusual. The problem is that this kind of knowledge might not be found in books, journals, or on the web, or if it exists, is scattered across many different web sites or buried in published papers. For example, some good general guidelines and procedures are available on the web, notably at the SEI's COTS based systems web site (<http://www.sei.cmu.edu/cbs/>) but Joe knows that his results would be

stronger if corroborated by specific previous experiences. Joe would thus have difficulty finding any good sources for this information in a timely manner.

## 2. CeBASE COTS Lessons-Learned Repository

One approach to facilitating knowledge sharing within communities is building experience repositories. The development of a COTS Lessons Learned Repository (CLLR), in which all authors are involved, is part of the Center for Empirically-Based Software Engineering (CeBASE) initiative. CeBASE ([www.cebase.org](http://www.cebase.org)) is an National Science Foundation (NSF) sponsored partnership between universities, research centers, and industry, whose main purpose is to promote experience sharing among software development practitioners. We present here the results of the effort focused on acquisition and dissemination of experience throughout the CBS community of interest. The participation of project managers, developers, COTS integrators, and other practitioners is essential in order for the entire community to benefit from this effort.

The CLLR is a living repository that can be accessed through <http://fc-md.umd.edu/ll/index.asp>. It is available for practitioners to use, comment on, add to, and help define its evolution. The repository will not grow or thrive unless it

meets the needs of the practitioners' community. So its actual evolutionary path will depend completely on how it is used and the needs expressed by its users.

In this paper we present the current status and capabilities of the repository, and our vision for its future evolution. We strongly encourage readers to comment on our ideas.

## 2.1 Repository Content and Organization

*Luckily, Joe finds a reference to the CeBASE COTS Lessons Learned Repository at <http://fc-md.umd.edu/ll/index.asp>. After accessing the repository and searching for lessons learned on "COTS evaluation", he finds several that catch his eye. One says "on safety-critical systems, only use components from a very mature vendor". Another says "using components from vendors who have been in business less than 3 years incurs a high risk of having to replace that component during maintenance, or even before initial deployment of the system". Since Joe's system is both critical, and is envisioned to have a long life, both of these lessons learned seem relevant. This gives Joe ideas for evaluation criteria. It also makes Joe think that, in addition to evaluating the components on his list, he really should be evaluating the vendors as well. So he decides to go to his company's software acquisition office and talk to folks there about their previous experiences with the vendors of components on his short list.*

*While talking to his company's procurement people, Joe got an earful. In addition to information*

*about companies and their maturity, he heard lots of horror stories. These included stories of broken contracts, non-responsiveness of help desks, low component quality, lack of documentation, vaporware, and sleazy negotiating tactics. For example, he found out that the vendors of the three most attractive-sounding products used terms such as "in alpha testing" as a synonym for vaporware. Based on all this information, he was easily able to filter out most of the 10 products on his short list, ending up with one final candidate. Because he had collected information on the product and the vendor, he was confident in recommending this one remaining product to his project manager.*

*Based on this experience, Joe recommended to his manager that all COTS-based development projects take into account this kind of vendor information in COTS evaluation. His recommendation was adopted and written into his company's procedures for COTS selection.*

*For justifying his COTS selection decision, Joe told his manager about the repository. It so happens that Joe's manager likes new gadgets, so he went online, browsed through the repository and was pleasantly surprised to find a few good pieces of advice, such as "if your customer does not have flexible requirements, using COTS is not a good option" and "get your customer involved early". That made him immediately initiate discussions and requirements negotiation with the customer.*

*Also, while talking to the acquisition people, Joe mentioned the lessons learned repository. They were curious about whether there was anything relevant to them there. Since it only takes a few clicks, they decided to give it a try. They learned about a project that had a problem with a multi-vendor solicitation that could have been avoided if a pre-award hearing would have been held. In order to prevent such a situation, Joe's organization decided to hold a hearing themselves.*

The lessons learned repository that supported Joe in his decision making is real and is currently seeded with an initial set of about 70 lessons learned extracted from the literature. The sources that provided these lessons include journal articles [3], workshop presentations [6], and government reports [1], [7]. In addition, we organize online workshops (eWorkshops [4]) and use these discussions to synthesize new lessons and to refine the existing ones. We are also in the process of consolidating our repository with lessons learned by the SEI that have been collected from a large number of case studies, but have not been published.

The lessons are described in the repository by a set of attributes, the most important being the ones describing the context in which the lesson was learned (type of system, type of company, number and type of COTS). Other attributes refer to type of data, recommended audience, relevant life cycle phase, etc. Most of the attributes were chosen based on a bottom-up effort to differentiate the lessons learned

in the initial repository, but others were added simply because they seemed to reflect issues of interest to potential practitioner users (e.g. impact on cost, quality and schedule).

The interface to the repository supports search and retrieval based on text searches over all attributes. Links to the original source of each lesson are also provided.

The repository also has a built-in facility for tracking various metrics related to repository usage. This metrics data can be used to tune the repository based on patterns of usage.

## 2.2 Capabilities/Features

Figure 1 presents the main components of the COTS Lessons Learned repository and how it is used.

The main component currently implemented is the COTS Lessons Learned database, which is accessible over the web, through any browser at <http://fc-md.umd.edu/ll/index.asp>.

The main component currently available is the COTS Lessons Learned repository. *Users* can interact directly with the repository and browse or search and retrieve lessons. They can also submit feedback about existing lessons or submit new lessons. The feedback and new lessons will go first to a buffer; they will be examined and validated by the *validator* and then uploaded. An administrator takes care of maintaining the repository and an *analyst* is responsible for the repository's evolution. A component of the system (currently under development) will allow dialogues

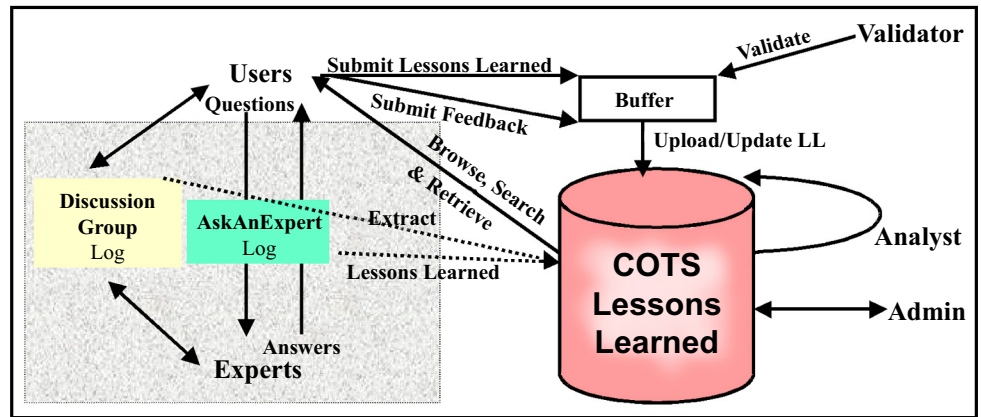


Figure 1. The COTS Lessons-Learned Repository Usage

Lesson Learned Statement	Recommended action	Details
Identify desired quality attributes and assess COTS against them	Assess desired COTS attributes early	<a href="#">Click here for details</a>
If you have a safety-critical system, you don't want state of the art COTS; you want mature components	Choose mature COTS	<a href="#">Click here for details</a>
You have to constantly monitor the state of the COTS components	Constantly monitor the state of the COTS components (and the market)	<a href="#">Click here for details</a>
Increased reliance on commercial items implies a different paradigm of system acquisition	Cooperation is needed among the program office, the stakeholders, the contractor, and the vendor. Simultaneous definition and tradeoff for system context, marketplace, and architecture and design	<a href="#">Click here for details</a>
If you can be flexible, COTS is cheaper. If not, it's more expensive	Don't go COTS if you can't bend your requirements	<a href="#">Click here for details</a>
Very few components worked as advertised	Evaluate the product in operational demonstrations. Involve users	<a href="#">Click here for details</a>
There are non-trivial adaptive maintenance costs if the product development is inconsistent with COTS Market evolution	Evaluate vendor's current evolution patterns for consistency with the developer's	<a href="#">Click here for details</a>
For large, multi vendor solicitations: hold pre-award hearings so that each vendor will have an opportunity to ask questions and all vendors will hear the same response	Hold pre-award hearings so that each vendor will have an opportunity to ask questions and all vendors will hear the same response	<a href="#">Click here for details</a>
Never pay the vendor to modify a COTS product	If the vendor does modifications for you, make sure the modifications are included in the product and effectively supported	<a href="#">Click here for details</a>

Figure 2. Search Result Example

Type	Lesson Learned Statement	Issue/Risk factor	Impact on cost	Impact on quality	Impact on schedule	Type of data	Recommended action	Comments/Story	Lifecycle Phase	Recommended audience	Aspect	Object	Type of system	Type of company	Number of COTS per project	Type of COTS	Date Created	Date added	Name	Organization	Email	Reference	Link
Good practice	If you have a safety-critical system, you don't want state of the art COTS; you want mature components	Ensure Safety	Unknown	Safety	Unknown	Qualitative	Choose mature COTS	16 projects from FAA, Army & Navy	Unknown	Project manager	Managerial/Technical/Market	Product vendor	Safety critical	Military	1-150	DBMS/OS/device drivers/network software	10/1/00	11/29/01	Patrick Lewis; Patrick Hyle; Marian Parrington; Elizabeth Clark; Barry Boehm; Christopher Abbs; Robert Manners; John Brackett	USC, SERC	boehm@personal.usc.edu	Lessons Learned in Developing Commercial Off-the-Shelf (COTS) Intensive Software Systems	

Figure 3. Lesson Learned Details Example

between users and experts, to support helping concrete problems. The logs of these dialogues are captured and used for extracting new lessons (as discussed in Section 3: Future Work).

A user can browse through the lessons or search for specific ones as shown in Figure 2. The statement of the lesson and the action recommended are listed on the front page thus immediately visible to the user. Details about a specific lesson are just a click away. The “Click here for details” link displays (as shown in Figure 3) all the values of that lesson’s attributes.

Feedback is always welcome and users can at any time provide feedback on any of the lessons learned. This feedback reflects the utility of the lesson to the user, and the user’s opinion about the applicability of the lesson in their specific context. An example of useful feedback is “this works differently in my environment because...” or “I experienced the same situation in a similar project”, or “I experienced the same situation in this different context ...”.

Users can submit feedback not only about individual lessons, but also about the repository itself, its technology, organization, and usage. This will allow the repository to evolve in the direction desired by its users.

The users are of course highly encouraged to contribute to the “community experience” with lessons they have learned themselves. In order to share this experience with their peers, developers and managers are asked to submit new lessons, by using an online submission form.

Data & Analysis Center for Software (DACS)

For guidance on the use of the repository, there is a set of frequently asked questions (FAQ) accessible from the main page. If an answer cannot be found in the FAQ, the user can submit a question and one of the tech support people will provide an answer. The new question-answer pair will be posted in the FAQ adding to the community knowledge.

Another capability of the system is a Visual Query Interface (VQI) [5] to the lessons learned, as shown in Figure 4. By using this VQI the user can visualize the entire content of the repository, which can facilitate the search for relevant lessons. In VQI, each dot displayed in the main window corresponds to one lesson in the repository. The set of lessons displayed can be filtered using the attributes as shown in the right window. On the X and Y-axes the user can select the attributes by which lessons should be displayed. In Figure 4, the X-axis represents the “Life cycle phase/activity” (e.g., COTS evaluation, COTS upgrade, Early development phases, Procurement, System acquisition) during which the lesson was learned

or can be applied. The Y-axis represents the “Object” of the lesson (e.g., Product, Process, People, Vendor). The lessons can be colored by a selected attribute - which in Figure 4 is “Recommended audience” (e.g. Program Manager, Project Manager, Developer). By clicking on a particular dot, details about the lesson will be displayed in a new window similar to the one in Figure 3.

## 2.3 Repository evolution

A year later, after his project had successfully completed, with the inclusion of the component he had recommended, Joe went back and took a look at his original “short list” of 10 candidate components. Almost all of the products he had eliminated were currently either no longer supported, or the vendors had gone out of business. Joe gathered information from other COTS-based projects that had taken place over the previous year and found many similar experiences. Joe felt this was a very useful lesson learned, so he went again to the CeBASE COTS Lessons Learned

Repository, but this time to add his own experience. He summarized his lesson as “information about component vendors is as important as information about components in the COTS

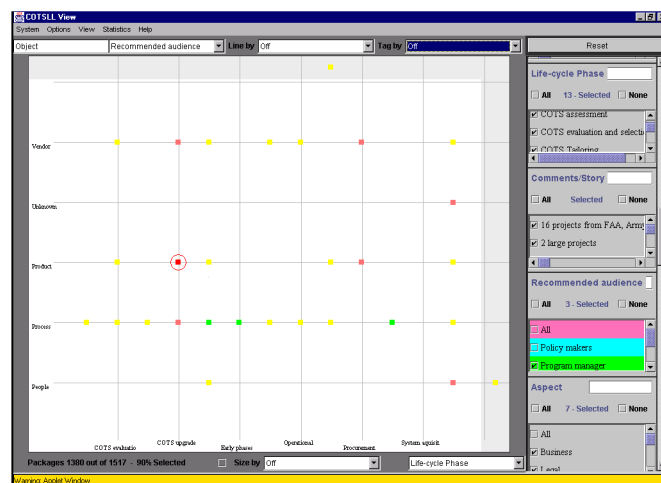


Figure 4. Using the Visual Query Interface (VQI)

article continued on page 20

# Did We Lose Our Religion?

by Lloyd K. Mosemann II, Science Applications International Corporation (SAIC)

## 1. Background

Back in 1990 I declared that the 1980s were a lost decade from the perspective of software development progress. The question I posed was: “*Will there be a breakthrough in the 1990’s?*” I went on to say:

*“It won’t happen automatically; people are too satisfied with unsatisfactory ways. We dare not make the mistake of complacency a la the automobile industry; we must push awareness and resource commitment to get ahead of the power curve of demand.”*

In 1994, I closed the annual DoD Software Technology Conference with the observation that the underlying need within the defense community is for **predictability**.

*“From a Pentagon perspective, it is not the fact that software costs are growing annually and consuming more and more of our defense dollars that worries us. Nor is it the fact that our weapon systems and commanders are becoming more and more reliant on software to perform their mission. Our inability to predict how much a software system will cost, when it will be operational, and whether or not it will satisfy user requirements, is the major concern. What our senior managers and DoD leaders want most from us, is to deliver on our promises. They want systems that are on-time, within budget, that satisfy user requirements, and are reliable.”*

The Question I should like to briefly explore in this article is:

“*Where are we today, and where will we be tomorrow?*” Did we lose our religion?

## 2. Did We Lose Our Religion?

Why do I use the metaphor of “religion”? Primarily because “religion” is the traditional example of “faith-based” behavior - that is, behavior that is based on a belief-system rather than on externally imposed rules such as the “law of gravity” or “she that has the gold, rules”. Remember: the emotional discussions regarding whether Ada or C++ should be preferred were frequently described as “religious” arguments based on beliefs rather than facts.

Sadly, I still see the world of software being governed by religious-like belief systems rather than objective appraisal. When I left the Pentagon six years ago I described some of what was happening as “bumper sticker” management, and the situation has not changed for the better. For example, “Use Best Commercial Practice”; or, “Buy Product not Process”.

What’s wrong with “best commercial practice”? Fact is, it just doesn’t exist among DoD contractors. It’s a fantasy created by those who want to streamline acquisition, making it possible to cut the number of oversight personnel by reducing the opportunity for oversight. The best way to justify a “hands off” attitude is to assert that contractors always do everything right!

There are more mature software organizations today. Virtually every large DoD contractor can boast at least one organization at SEI CMM Level 4 or above, and several organizations at SEI CMM Level 3. On the other hand, most DoD software is still being developed in less mature organizations - mainly because the Program Executive Officer (PEO) or Program Manager doesn’t demand that the part of the company that will actually build the software be Level 3!

Back in 1991 Paul Strassmann, who was then the Director of Defense Information, said:

*“The #1 priority of DoD, as I see it, is to convert its software technology capability from a cottage industry into a modern industrial method of production.”*

Guess what? That has not happened. Why not? Because this requires “software engineering”. Software engineering encompasses a set of three key elements — methods, tools, and procedures — that enable the manager to control the process of software development and provide the practitioner with a foundation for building high quality software in a productive manner.

The fundamental ingredient in a “software engineering” approach is the design of a robust software architecture. By architecture I don’t refer to the grouping and linkage of servers and routers and PCs, but rather to the inherent design of the software itself - the identity of modules and their relationships, including in particular the



I am told that SEI is still being asked to do Independent Technical Assessments of “Why a software acquisition has not produced the desired working software

infrastructure, control and data interfaces that permit software components to operate as a system. I am told by an attendee at a DoD Systems Design Review several months ago that the contractor described his proposed system as “modular”. That’s a good architectural term, and it was accepted at face value. In fact, the system, when looked at by the independent attendee, was found to have only two modules. When this was brought to the Government Program Manager’s attention, he said, “The contractor says it is modular. He’s smarter than we are.” This little incident underscores two facts: architecture was understood by neither the contractor nor the government PM, and the probability of a successful software delivery is low.

All too often the DoD excuse for not requiring an architectural evaluation is that “requirements change rapidly — can’t afford to be locked into a specific architecture”. Wrong - that is the very reason that attention should be given to architecture, so that changes to requirements can be accommodated easily.

system?” Why wait to ask SEI to come in to do a post mortem and tell you how you screwed up? Why not ask them to come in, first, to review the RFP and Statement of Work, and, second, to assist in evaluation of the software engineering practices, software development environment, and proposed architecture proposed in response to the RFP, and then, again, after award to assess the quality of the software engineering and development process? SEI isn’t cheap, but terminating a \$100M project for lack of satisfactory performance isn’t cheap either.

Interestingly, when one thinks about “best commercial practice”, there are two very different worlds out there. There is the government contractor world; and, there is the real commercial world - banks, insurance companies, UPS and FedEx, Eckerd Drug, and Disney World. These companies develop their own software using the best available software tools such as Rational’s software development environment. They don’t pick the cheapest tools. They don’t rely on COTS or outside software developers - their software is their

business, and they consider that it provides them a competitive advantage, and they want to control it, and they use the best tools available regardless of cost.

Product Line Developments are also becoming increasingly commonplace in the true commercial world. In addition to the original example (CelsiusTech, the Swedish firm that was the first to exploit the benefits of a product line architecture approach to software application development back in the late 1980s), there are now a number of firms that you have heard of who are using an architecture-centered approach: for example, Nokia, Motorola, Hewlett-Packard, Philips, Cummins Engine, and (believe it or not) one government application at the National Reconnaissance Office (NRO). The NRO is enjoying 50% reductions in overall cost and schedule, and nearly tenfold reductions in defects and development personnel. Let me list for you the most common and relevant-to-DoD reasons that these companies give for adopting the “Architecture-Centered Product Line Approach” to software development:

- Large-scale productivity gains
- Improved time-to-market = field weapons faster
- Improved product quality
- Increased customer satisfaction ~ Warfighter satisfaction
- Enables mass customization
- Compensates for inability to hire software engineers

These companies and the NRO do have “best practices”, but the “best practices” are not yet widely recognized as such. Frankly, it will take DoD PEOs (not Program Managers) and their overseers, the Service Acquisition Executives, and especially the DoD Comptroller and PA&E folks, to recognize and direct the product line approach to software development and acquisition. (Unfortunately, these folks are not known for their software acumen.) The major impediment to the Product Line Development approach, aside from ignorance of its benefits, are cultural, organizational, and, especially, the DoD’s stovepipe approach to budgeting and funding. DoD has many potential software product lines. None of them have been built, largely for “political” and stovepipe budgeting reasons. As a result development and fielding of defense systems continues to take longer, cost more, and lack predictable quality. Product lines require strategic investment, which appears to be outside the DoD Comptroller and Acquisition communities’ frames-of-reference. Yet, it is the Department of Defense that most often uses the term “strategic”.

Cummins’ Engine Company used to take a year or more to bring software for a new engine to the test lab - now they can do it in less than a week! I strongly recommend that you obtain a copy of a new book, *Software Product Lines*, just published in 2002 by Addison-Wesley. The authors are from SEI. Sadly, with the exception of the NRO, it appears that the readers are mainly from commercial organizations. There was a

previous version of the book, *Software Architecture in Practice*, that sold more than 14,000 copies, but I doubt that few, if any, DoD folks have read it.

I work for one, but let me tell you that, although government contractors are commercial organizations, they do not have an identifiable “best commercial practice”. They basically will provide what the Government asks for. The only reason many of them can now boast of having at least a few SEI maturity level organizations is because, starting 10 years ago, many government RFPs required a Software Capability Evaluation as a condition of bidding. In fact, sad to say, many contractors today put into their proposals satisfactory CMM credentials, but then actually perform the work with an organization who couldn’t spell CMM. Why does the Government let this happen? Why aren’t there Software Capability Evaluations anymore? - the word is, they “take too long and cost too much”. Better to make a quick award, and then, down the line, accept an inferior product or terminate for convenience. Too often what the government has been asking for is COTS. How many failures of COTS-based acquisitions have there been over the past decade? Too many!

“Best commercial practice” is not eliminating all software smarts in government and relying 100% on contractors to deliver good software. “Best commercial practice” is what real commercial companies are doing. They have in-house software expertise, they use a robust software development

environment, and they base their software development on a sound software architecture. It is no secret/no surprise that Rational and their competitors have a growing market in the commercial world and a shrinking market in the government world. I’m not suggesting that the Government can or should develop software in-house. I am strongly suggesting that the Government needs enough in-house software expertise to know what it is buying. It is still true that you get what you pay for”, and that “apples and oranges” are not the same!

Watts Humphrey recently published a new book entitled *Winning with Software - An Executive Strategy*. Not surprisingly, the book is directed primarily at executives of commercial companies. But every DoD acquisition executive, PEO and Program Manager needs to read and understand its message. Frankly, its message is pretty simple: *Software projects rarely fail for technical reasons; invariably, the problem is poor management*. He poses two interesting questions and buttresses them with numerous examples:

1. “Why do competent software professionals agree to dates when they have no idea how to meet them?”
2. “Why do executives accept schedule commitments when the engineers offer no evidence that they can meet these commitments?”

He asserts that Management’s undisciplined approach to commitments contributes to every one of the five most common causes of project failure:

1. Unrealistic schedules
2. Inappropriate staffing
3. Changing requirements
4. Poor quality work
5. Believing in magic.

### 3. What is Formal Methods Programming?

Basically, it is all of the above rolled together: sound management, established engineering processes, robust software development environment, model based architecture, and a reliable programming language. I was thrilled to see, in the March edition of *CROSSTALK*, an article by Peter Amey of Praxis Critical Systems, in which he stated:

*“There is now compelling evidence that development methods that focus on bug prevention rather than bug detection can both raise quality and save time and money.”*

He then goes on to say that a key ingredient is the use of unambiguous programming languages that allow rigorous analysis very early in the development process. I was an early and vocal advocate of Ada, primarily because, unlike other languages, it enforces basic software engineering practice. In that article Peter describes the use of a subset of Ada known as SPARK. He says,

*“The exact semantics of SPARK require software writers to think carefully and express themselves clearly; any lack of precision is ruthlessly exposed by its support tool, the SPARK Examiner.”*

He indicates that there were significant savings from using

SPARK in a critical avionics program, including most notably that formal FAA test costs were reduced by 80%. Unfortunately, this is an isolated DoD example - the same rigor and discipline of formal methods programming should apply for all major software intensive system developments.

### 4. Predictability

Finally, let me say a word about Predictability. Predictability is the only metric that the warfighters care about. Question is - how can we make the warfighters (including the PEOs and PMs who are charged with delivering the capabilities needed by the warfighters) smart enough to know that you can't just buy software as a product off the showroom floor. There must be understanding of the software engineering paradigm that produces software. In fact, more than this, to be assured of getting software that works on a predictable schedule and at predictable cost requires that someone in government be smart enough to enunciate the basic processes that will be employed by the contractor to produce. This is important because, otherwise, competitors will bid an unrealistically low price and unrealistically fast schedule, and be awarded the contract. To perform at the low cost means no robust software development environment, no time and effort devoted to creating a valid software architecture, and probably means cheap people. If the government wants to “get software right”, sufficient process guidance must be given to assure that the contractors all bid essentially the same level of capability. I believe that

Government should be explicit about the need for architecture, a robust software development environment, perhaps even the requirement for a language like SPARK, but, as a minimum, it needs to specify that the performing organization must be at least CMM Level 3. Why? Because at Level 1 virtually all projects have cost and schedule overruns, whereas at Level 3 virtually all projects are on target. As regards Defect Rates and \$ per Source Line of Code, there is substantial improvement on the order of 20-50%. It really is true that YOU GET WHAT YOU PAY FOR. If you want it cheap, you'll get it cheap - but it may not work in the manner envisioned, if it will work at all.

Where have we been? The Government I fear has been wallowing in a slue of self-deceit, thinking it need not have software expertise, but can simply rely on the so-called “best commercial practices” of contractors. The problem is that, left to their own devices, the “best practices” of defense contractors are not very good. Are there best commercial practices? You bet! The banks, insurance companies and other truly commercial enterprises have them. But they have them, not because of some automatic external happenstance, but because their senior managers have had the moxy to realize that it takes money to make money, and that it takes software expertise to develop or acquire software. The Government needs to go and do likewise. Otherwise, the decade of 2000 will likely not show any lessening of the Software Crisis that has carried over from the 1990s.

# Habitats: Toward An Architecture for the Global Information Grid

John Salasin, Defense Advanced Research Projects Agency (DARPA),  
Assad Moini, Software Productivity Consortium, and Gwen Williams, Schafer Corporation

## 1. Introduction

The military's vision of future warfare as described by Joint Vision 2010 (JV2010) emphasizes broad use of advanced Information Technologies (IT) to significantly improve traditional military capabilities. It implies tight integration and interoperability among DoD systems both horizontally across different domains (e.g., intelligence, operations, and logistics) and vertically between command levels (e.g., National Command Authority, Commander-in-Chief, Joint Task Force Commander, and Small Unit Operations). This tight integration will also include dynamic multi-national, multi-echelon teams engaged in continuous (joint) collaborative planning and/or engagement.

JV2010 also stresses the *importance of information superiority and decision superiority* as the key to the US military superiority in the 21st century. Decision superiority is the state of having the ability to collect, fuse, process and disseminate an uninterrupted flow of information and command, while denying an adversary the ability to do the same. Of course, information superiority is not an end to itself but a means to achieving decision superiority. The objective is to leverage and transform information superiority into a state where better decisions are made. The overwhelming theme in all of this is the enabling of both greater

information sharing and effective, fluid collaboration among heterogeneous units, organizations and systems as dictated by the evolving mission needs. Achieving and maintaining information superiority places a huge emphasis on operational application of information technology. As a result, our success in future operations will heavily depend on our ability to rapidly acquire, assimilate and leverage emerging information technologies.

To address the JV2010 needs, several DoD programs are currently underway including the Navy's Network-Centric Warfare (NCW), the Air Force Joint Battlespace Infosphere (JBI), and the Army's Future Combat Systems (FCS). All these programs share and rely on a common construct called the Global Information Grid (GIG), an Internet-like network of networks, providing the necessary substrate for connecting sensors, shooters, commanders, analysts, legacy and future command and control (C2) and warfighting assets. The GIG serves two main functions. First, it is a notional construct (a conceptual model) for presenting, evaluating, calibrating and unifying advanced military research and technology planning required to realize the JV2010 vision. Second, it is the physical infrastructure that will realize the goal of migrating the DoD warfighting enterprise toward a globally integrated command and control platform, supporting a broad

spectrum of future warfighting needs, including operations other than war. Much of the previous work on the GIG has focused on connectivity, networking and basic interoperability issues, such as how to incorporate airborne nodes into the GIG and integration of newer, IP-based networking backbone and the legacy, military radio communication protocols. Among the areas yet to be addressed, are the issues revolving around the control, allocation, monitoring and management of the GIG resources.

We are proposing *habitats* as a unifying systems concept and an architectural construct for identifying, formulating, assessing and presenting issues, concepts, engineering principles, methods and solution approaches required to make the Global Information Grid a reality. The *habitats* concept also serves as a test bed for ideas and a focal point for integration of a broad range of research issues into a cohesive research agenda.

The next two sections briefly characterize the GIG environment and the key issue of control and management of the GIG resources, respectively.

## 2. Characterizing the Global Information Grid

The GIG has been characterized as an information environment providing value-added services supporting uninterrupted, secure flow of mission-critical planning

and survival information in a globally disbursed environment among producers and consumers. In this respect, the GIG is more than just a network of networks; it is a *system of systems* consisting of both physical and logical components. As an architectural construct, the GIG promotes a service-centric approach as opposed to the earlier DoD application centric platforms.

To be effective, the GIG must provide the following capabilities:

- Serve as a generic, plug-and-play infrastructure, accommodating rapid, ad-hoc adaptation and reconfigurability to support evolving mission needs, for a broad spectrum of operations including joint, coalition as well as operations other than war
- Provide a seamless environment for end-to-end access to, fusion, dissemination, and presentation of information services regardless of the providers' underlying platforms or networks
- Provide the required computing resources (connectivity, bandwidth, processing) on demand as well as on an anticipatory basis
- Provide management and control mechanisms for provisioning both local and global resources
- Allow for easy extensibility with respect to infusion and insertion of new technologies, while maintaining backward compatibility with the legacy and heritage systems as well as

the existing suite of military-unique networking and communication protocols

Implementing the GIG requires dealing with numerous technical challenges especially with regard to end-to-end interoperability and security, as well as organizational issues related to the ownership, control and management of assets and resources linked by the GIG. From a purely technical standpoint, building a global information infrastructure without capitalizing on currently available commercial infrastructure, such as the commercial Internet and Web technologies, is not feasible. In fact, adopting and leveraging these existing commercial technologies in building the GIG is a financial and time-to-market imperative. However, because commercial technologies are designed with a different set of priorities, a number of issues must be addressed.

During this decade, Moore's Law projects a 32-fold increase in processing power and a 256-fold increase in storage capacity; all at the same cost as today's leading edge commercial technology. During the same period, advances in optical networking and wireless technology are expected to produce a dramatic increase in networking bandwidth and connectivity at considerably affordable prices. These technological advancements raise two issues. First, the number of entities situated on and linked by the GIG can potentially be in the billions, making it nearly impossible to monitor, control or manage as a single domain. Second, much of this enabling

technology is and will be proliferated throughout the world, allowing potential adversaries to achieve a greater level of sophistication in their military capabilities. A key challenge for the GIG architecture is the ability to rapidly accommodate, assimilate and co-evolve with the emerging commercial technologies.

### **3. Managing and Organizing the GIG Resources**

The GIG must support uninterrupted flow of mission-critical planning and survival information. The highly decentralized nature of the GIG raises serious concerns with respect to the timely and predictable allocation and recruitment of geographically dispersed resources in a manner that is consistent with evolving operational needs and priorities.

The ability to dynamically discover, authenticate and allocate GIG resources will require new technologies. Decentralized control and management of heterogeneous resources, owned or controlled by multiple authorities, over terrestrial, wireless, radio and satellite networks raises serious issues with respect to authorization, mutual authentication, and coordination for access. Additionally, each local authority may have its own internal policies, checks and balances, which are implemented as a complex hierarchical decision-making process. This makes both finding and securing the required resources even more difficult. Development of authority and

accountability technologies would facilitate acquisition and authentication of resources.

## 4. Introducing Habitats

Historically, military systems (e.g., command and control) have been built as a collection of disparate, unique platforms with fixed functionality, intended to address specific operational needs. Many of the existing DoD communication systems are vertically integrated to satisfy specific warfighting requirements. Many legacy systems have built-in, military unique communication equipments. Achieving interoperability and information sharing among these systems, even at a limited level is not simple.

Habitats are computational building blocks for arranging and managing dynamic relationships in rapidly changing, decentralized environments, such as the GIG. Habitats provide a computational model for organizing, coordinating and managing (human-to-human, human-to-systems or system-to-system) interactions in environments where independent actors must heavily rely on information and communication infrastructures to arrange and coordinate interdependent (joint) activities. These activities may typically cross multiple organizational and functional boundaries. As such, habitats provide the necessary shared context for integrating business processes across independent organizations.

Conceptually, habitats resemble virtual (human) organizations, dynamically formed to augment or

extend human organizations in support of a particular business activity or mission. Therefore, habitats can be either mission-centric or organization-centric. They serve to provide a shared context and transactional boundary for ongoing activities involving humans, agents and systems. Habitats also serve as shared collaboration spaces mediating secure, reliable and transparent access to remote C2 and warfighting assets, irrespective of their (network) location or underlying platform, in a manner consistent with organizational policies and procedures and subject to appropriate checks and balances.

Habitats exploit the power of networks to bring together geographically dispersed, distributed entities (human actors, warfighting assets, C2 infrastructure...) toward accomplishing a specific task or mission. As such, they go beyond merely facilitating information sharing; they serve as a conduit, arranging new forms of interaction and collaborative relationships.

The goal of habitats science and technology is to provide the necessary technical infrastructure needed to ensure that independent groups or organizations, from distinct spheres of powers, can effectively and expeditiously align their activities to achieve a common objective. Toward this end, the habitats technology must address organizational, operational and technical issues that arise in the context of ad-hoc coordination, collaboration and interoperability among otherwise independent organizations and systems.

Habitats also serve to provide a computational context and mechanism for partitioning and scoping activities and automated (business) processes that span multiple systems and organizational boundaries. As such, they provide transactional boundaries necessary for compartmentalizing, monitoring, auditing, or constraining activities that may span heterogeneous platforms and networks. They also provide a unified computing model that combine and subsumes the traditional distributed objects and hypermedia web document model.

The following section illustrates the notion of habitats in the context of a combat situation, using an operational scenario. Today, much of the required collaboration is complicated and hampered by the fact that many of the functional units and systems are often spatially dispersed and have hard-wired communication line (stove piped). The concept of habitats is used to dynamically organize otherwise stand-alone systems into a collaborative, networked system of systems.

## 5. Habitats Desiderata

In this section we present a set of desiderata (necessary characteristics) that must be fulfilled by the future habitats technology. To meet the following desiderata, the habitats technology must provide universal (platform and operating-system independent) supporting mechanisms at the GIG infrastructure level.

- **Dynamic Membership Model.** New members are admitted or enlisted and old ones can leave

the habitats (voluntarily or become disconnected) or their membership can be revoked if their integrity is compromised. Traditional networks generally rely on the existence of a permanent, centrally managed security service. In habitats, the entire functionality associated with security is fully distributed across all the members. This is necessary to ensure a high degree of resilience to unexpected (communication) failures or removal of a member. The key elements of the habitats dynamic membership model are: the mutual authentication of habitat members, proof of habitat membership to outsiders, and authentication of members with special privileges. The dynamic membership model is required for incorporating or removing entities (e.g., coalition partners, non-DoD, civilian agencies, or allies). Habitats must support flexible and interoperable policy management and enforcement by providing a dynamic membership model. Membership in secure habitats may be required to facilitate total resource management as well as access to data.

- **Dynamic, Granular Trust Model.** Military organizations often need to cooperate or interoperate with civilian or non-governmental organizations or members of military alliances toward a common goal. In many cases, it may be necessary to share or transfer highly sensitive information, for example by providing access to intelligence or reconnaissance assets on a

temporary basis subject to limited, asymmetric trust. Additionally, given the evolving nature of operations and alliances, the level and granularity of trust among the partners may change rapidly. Habitats must explicitly support the notion of granular trust relationships (coarse and fine grain trust levels) as well as an asymmetric trust model. Habitats infrastructure must provide the necessary mechanism to ensure end-to-end authority accountability across interacting, overlapping habitats.

- **Dynamic Policy Management.** Coordination policies highly depend on the context of the activities and the nature of collaboration. Even within the same context, policies often may have to evolve due to the changing nature of the collaboration itself (with respect to both scope and extent). In habitat relationships, for example in a coalition or operations other than war (OOTW) setting, the level of trust among participants (humans, military and non-governmental organizations) can change over time, thus necessitating co-synchronization or updating policies within and across interacting habitats.
- **Boundlessly Scalable.** Scalability is the ability to increase or decrease the size and/or scope, as necessary, to accommodate dynamic changes in operations, caused by the changes in the environment. Habitats will be capable of

expanding (upsizing) and retracting (downsizing) bandwidth and internal processing capacity relative to size, scope and quality of service. Scaling up is the ability to dynamically expand or extend operations as the workload grows. Scaling down is the ability to contract operations in size or scope, for example by shedding functionality or resources, in response to diminishing demands, under consumption of resources or simply to cope with independent local or remote failures. Expandability and adaptability are two complementary aspects of scalability.

- **Expandable.** Expandability refers to a change in size or scope necessitating a change in structure or organization of habitats. Expandability in habitats requires the ability to create new habitats practically anywhere, anytime. New habitats can be bootstrapped from existing ones, for example, by federating existing ones. At the same time, existing habitats must be able to incorporate new habitats within themselves and be able to create new (sub-) habitats by reorganizing, combining or regrouping.
- **Adaptable.** Adaptability is the ability to evolve habitats structure (roles, responsibilities, authority and accountability relationships) in response to the evolving mission needs caused by external factors, while meeting commitments in place

(e.g., carrying out activities already in progress). Adaptability demands reorganization by amending or extending existing habitats by incorporating new entities and/or assets.

- **Transitory in Nature.** Habitats are dynamic. Once formed, they continue to co-evolve with changing mission needs and objectives. They are ultimately dissolved or placed in a state of hibernation; habitats placed in hibernation can be resurrected and placed back into the operational mode as the need arises. In this respect, habitats are organizational/operational blueprints for rapid creation of electronic organizations (rapid strike teams/cells) required to support specific mission needs.
- **Overlapping.** Habitats can overlap in space and time. Two habitats can overlap in space by virtue of sharing common entities or assets (e.g., the same ground sensor or airborne node can be concurrently assigned to multiple habitats, subject to different access and usage rules). Two habitats can temporally overlap because of their mutual dependence on completion status of events or activities hosted by another habitat.
- **Managing Quality Of Service.** Habitats infrastructure must provide required computing resources (connectivity, bandwidth, processing) on a just in-time, anticipatory, or reservation basis.

- **Unified Control and Management.** Habitats infrastructure must provide universal (platform and network independent) mechanisms for management and control of both local and global resources.

## 6. Time Critical Target: An Example

Consider a Time Critical Target (TCT) cell habitat. The function of the TCT cell is to assign resources to emerging high priority ground targets, identified by Joint-STARS (JSTARS). This habitat is instantiated (formed) when a Moving Target Indicator (MTI) track appears and JSTARS produces a track report. The Air Defense System Integrator System (ADSI) automatically assigns a System Track Number (STN) and begins reporting the track with updates to attributes such as location, speed, direction, and classification (e.g., friendly, enemy or undetermined).

The ADSI then evaluates the track and correlates JSTARS information with that from other intelligence sources, e.g. point of origin or communications intelligence (COMINT) reports on payload.

The TCT habitat is dynamic. Based on its location (e.g., the theatre of operation) it has the ability to interoperate with certain specific Information, Surveillance and Reconnaissance (ISR), surface-to-surface, Combat Operations, or Area Air Defense (assets) habitats. As a background task, it can update its awareness of and access to information about resources in its current area of concern (based on the JSTARS location). It is activated

on nomination of a dynamic target and hibernates when no targets are active (i.e., the Dynamic Target Queue is empty).

The TCT cell habitat provides specialized services to its components. These might include various models (e.g., dealing with munitions effectiveness, platform/resource performance, meteorological effects, and available bandwidth/latency). It provides access to specialized, context-dependent data (e.g., enemy order of battle). It coordinates components (e.g., resource location and status agents, time sensitive resource data update tools, tools for coordination and deconfliction). It also enforces various rules or constraints (e.g., conditions requiring different levels of authorization, allowable threshold of potential collateral damage, and required authorization), commits authority levels, required sequence of operations (process), rules for lawful targets (and when they need to be enforced), fault-tolerant fallbacks (QoS, data availability concerns), and access and update of authorization capabilities and permissions. All of these models, services, and rules are context sensitive – they may depend on the theatre involved or the state of hostilities.

Note that this scenario and its follow-on activities require dynamic recruitment of resources and services – a primary responsibility of the habitat. These resources may include, for example, information about potential second sensor sources and appropriate/available weaponry assets. This information might come from other (than Air

Force) services/organizations. It could include: sensor or weapon capabilities, current tasking/scheduled tasking, current location, time to image or reach target, tasking authority, time sensitivity of data, and weather.

The Table 1. illustrates the current situation and the manner in which habitats will address specific present needs and shortcomings.

## 7. Conclusion

Today, the U.S. military is in the process of transforming itself into a nimble, cohesive organization of the information age by replacing the rigid, stove piped warfighting infrastructure of the past with a more flexible, agile organizational structure equipped with distributed sensing, planning and execution capabilities, all networked together via the Global Information Grid. In this networked environment, the relationships and the resulting interactions are substantially different from the traditional hierarchical command and control model. To successfully operate in this environment, the traditional top-down C2 hierarchy must be modified to allow a more decentralized, peer-to-peer decision process at the lower levels, across units, echelons, organizational and functional boundaries. The habitats concept and its supporting technology will enable a new generation of systems that can successfully and predictably operate in a network-centric world of vastly distributed and dispersed resources, devices and users interacting via the GIG.

**Table 1: Benefits of Habitats**

Current Situation	Benefits (with Habitats)
<ul style="list-style-type: none"> <li>Lack of visibility about potential resources not under OPCON (Operational Control) of TCT organization.</li> </ul>	<ul style="list-style-type: none"> <li>“Business rules” embedded in the habitat and context information it provides to components offer: <ul style="list-style-type: none"> <li>Horizontal access to information</li> <li>Access to broader, richer set of assets</li> </ul> </li> </ul>
<ul style="list-style-type: none"> <li>Stovepipes required for decision approval (time delay, latency)</li> </ul>	<ul style="list-style-type: none"> <li>Improved coordination subject to checks &amp; balances (constraints defining allowable interactions of the habitat with other entities)</li> </ul>
<ul style="list-style-type: none"> <li>Making targeting decisions takes too long</li> </ul>	<ul style="list-style-type: none"> <li>Ease of incorporating business rules / constraints at habitat level of abstraction will provide quicker decisions, thereby reducing planning cycles from 24 to 4 hrs – by reporting exception only and involving experts rapidly</li> </ul>
<ul style="list-style-type: none"> <li>Ineffective utilization of resources due to poor coordination across services, functional organizations and/or units</li> <li>Lack of visibility into resources availability</li> <li>Inability to task resources not under OPCON. This is a major problem when resources are limited.</li> </ul>	<ul style="list-style-type: none"> <li>Better decisions due to business rules and context sensitivity: <ul style="list-style-type: none"> <li>Involvement of appropriate levels</li> <li>Involvement of “best experts”</li> <li>Better support to coordination, use of limited resources.</li> </ul> </li> </ul>
<ul style="list-style-type: none"> <li>Duplication of effort for systems development and maintenance</li> </ul>	<ul style="list-style-type: none"> <li>Reuse of components by abstracting context-sensitive processing/data to habitat level.</li> <li>Flexibility with respect to analysis components used (e.g. Army does best weapons system effectiveness for ground to ground, Air Force for air to ground)</li> </ul>
<ul style="list-style-type: none"> <li>“One size fits all” syndrome</li> <li>Same tools and systems must be equally applied to large-scale general warfighting operations as well as small conflicts or OOTW of actions</li> </ul>	<ul style="list-style-type: none"> <li>Habitats are tailorable (context sensitive) operations to different types (small group to theatre-wide conflict)</li> <li>Ability to form habitats rapidly to include nontraditional participants.</li> </ul>
<ul style="list-style-type: none"> <li>Incompatible “business processes” (e.g., rules of engagement) across services or functional organizations</li> </ul>	<ul style="list-style-type: none"> <li>Rules to specify process coordination points and negotiation protocols.</li> </ul>

## 1. Introduction

The 1990s was to be the decade of fault tolerant computing. Fault tolerant hardware was hot and software fault tolerance was about to happen. But it didn't. Instead, the Web Wave took off. Until a rash of server failures, denial of service attacks, web service failures and the September 11 tragedy we lost interest in fault tolerance. Even though Software Fault Tolerance was not hot, there was progress. One breakthrough is the use of Software Rejuvenation. Here are three ways it was used to greatly improve software reliability without having to do the pain staking work of digging out every defect even if they would never cause a system failure.

Software faults are most often caused by design faults. Design faults occur when a software engineer either misunderstands a specification or simply makes a mistake. Software faults are common for the simple reason that the complexity in modern systems is often pushed into the software part of the system. It is estimated that 60-90% of current computer errors are from software faults. Running software is a predictable state machine. Software manipulates variables that have states. Unfortunately flaws in the software that permit the variables to take on values outside of their intended operating limits often cause software failures.

Software rejuvenation is special software that gracefully terminates

an application, and immediately restarts it at a known, clean, internal state. Instead of running for a year, with all the mysteries that untried time expanses can harbor, a system is run for one day, 364 times. It is re-initialized each day, process by process, while the system continues to operate. Rejuvenation precedes failure, anticipates it, and avoids it. It transforms non-stationary, random processes into stationary ones.

## 2. Billing Data Collection

Two years of operation have passed with no reported outages for one system that collects billing data from telephone company switches. Its rejuvenation interval is set at one week. In another billing data subsystem a 16,000 line C program with notoriously leaky memory failed after 52 iterations. After adding seven lines of rejuvenation code with the period set at 15 iterations, the program ran flawlessly.

## 3. Store and Forward Message Switcher

While software cannot be designed without bugs, it does not have to be as buggy as it is. For example, as early as 1977, a software based store and forward message switching was in its fourth year of operation and it handled all administrative messages for Indiana Bell without a single failure. This record was achieved after a very buggy start followed by a substantial investment in failure

prevention and bug fixes. One particularly error-prone software subsystem was the pointers used to account for clashes in the hash function that indexed a message data file. The messages could remain in the file system for up to thirty days. There were many hash clashes due to the volume of messages and the similarity of their names. Once the obvious bugs were fixed the residual ones were hard to find. This led to unexpected behavior and system crashes. A firm requirement was not to lose any messages. Failures exhibited by latent faults can appear to be random and transient. But they are predictable if only we can get the initial conditions and transaction load to trigger them. They are sometimes called Heisenbugs. It was just too costly to find and fix all the Heisenbugs in the file index code, so rejuvenation was used to rebuild the hash tables daily in the early hours of the morning when there was no message traffic. With fresh hash tables, the chances of triggering a fault was small especially after the bugs that were sensitive to the traffic mix were found and fixed. This experience shows that it is not necessary for software to be inherently buggy.

## 4. NASA Uses It Too

The NASA mission to explore Pluto has a very long mission life of 12 years. A fault-tolerant environment incorporating on-board preventive maintenance is critical to maximize the reliability of a spacecraft in a deep-space mission. This is based on the inherent system redundancy

(the dual processor strings that perform spacecraft and scientific functions during encounter time). The two processor strings are scheduled to be on/off

duty periodically, in order to reduce the likelihood of system failure due to radiation damage and other reversible aging processes

Since the software is reinitialized when a string is powered on, switching between strings results in software rejuvenation. This avoids failures caused by potential error conditions accrued in the system environment such as memory leakage, unreleased file locks and data corruption. The implementation of this idea involves deliberately stopping the running program and cleaning its internal state through flushing buffers, garbage collection, reinitializing the internal kernel tables or, more thoroughly, rebooting the computer.

Such preventive maintenance procedures may result in appreciable system downtime. However, by exploiting the inherent hardware redundancy in this Pluto mission example, the performance cost is minimal. One of the strings



is always performing and starting it before the current active string is turned off can mask the overhead for a string's initialization. An essential issue in preventive maintenance is to determine the optimal interval between successive maintenance activities to balance the risk of system failure due to component fatigue or aging against that due to unsuccessful maintenance itself.<sup>1</sup>

## 5. Where To Get Rejuvenation

Watchd and Libft are software fault tolerance components. They may be used with any UNIX or NT application to let the application withstand faults.

Watchd is a watchdog daemon process for detecting UNIX process failures (crashes and hangs) and restarting those processes. The fault tolerance mechanism is based on a cyclic protocol. They may be obtained at the Lucent web site, [www.lucent.com](http://www.lucent.com).

Windows 95 has a special library WinFT that provides automatic detection and restarting of failed processes; diagnosing and rebooting of a malfunctioning or strangled OS; checking pointing and recovery of critical volatile data; and preventive actions, such as software rejuvenation. Joao Carreira et. al., "Fault Tolerance for Windows Applications," *Byte Magazine* February 1997, pp 51-52:

## 6. Wrap up

Most software runs non-periodically, which allows internal states to develop chaotically without bound. Software rejuvenation seeks to contain the execution domain by making it periodic. An application is gracefully terminated and immediately restarted at a known, clean, internal state. Failure is anticipated and avoided. Non-stationary, random processes are transformed into stationary ones. The software states would be re-initialized each day, process by process, while the system continued to operate. Increasing the rejuvenation period reduces the cost of downtime but increases overhead. Rejuvenation does not remove bugs; it merely avoids them with incredibly good effect.

Rejuvenation does not remove bugs that exist beyond its carefully circumscribed limits. Instead, it avoids the vast unknown territory that conceals them.

1. Y. Huang and C. M. R. Kintala, "Software Implemented Fault Tolerance: Technologies and Experience", *Proceedings of 23rd Intl. Symposium on Fault-Tolerant Computing*, Toulouse, France, pp. 2-9, June 1993;

Also appeared as a chapter in the book *Software Fault Tolerance*, M. Lyu (Ed.), John Wiley & Sons, March 1995.

evaluation process.” He also provided more information about the context of his observations (e.g. in defense-related, mission-critical systems) and provided a few more pieces of information, then submitted all this for others in the community to access.

The repository content is growing organically by contributions from users like Joe who add new lessons. The content also evolves as a result of analysis, synthesis, and refinement of the existing lessons. CBS experts, in collaboration with the maintainer of the database, perform this activity. The attributes used to characterize and classify the records will also evolve over time, again, in the direction where the use and the need will be.

*Based on his experience with his CCTwo project, Joe became a regular user of the CCLR. One day, as he was browsing for information, he came across a lesson that had been derived from the lesson he had earlier submitted, along with similar lessons. While his lesson had addressed the general importance of vendor information in COTS evaluation in his domain, this new, synthesized lesson went further. It said, “If your customer is willing to negotiate the requirements, it is cost-effective to choose components from more dependable vendors than to choose components with a better functional fit to the original requirements”. This statement was based on lessons submitted from software engineers in various domains who had both*

*good and bad experiences with vendors. Some of them had done detailed cost/benefit analyses. This gave Joe a lot more information about how to use vendor information on future COTS evaluations. It also gave him a good feeling to know that he had contributed in a meaningful way to the body of knowledge in his community.*

### 3. Future Work

*On another COTS-based development project, Joe ran into another problem: His manager had asked him to come up with an estimate for the amount of time it would take his team to integrate several COTS components into a subsystem of a large vehicle tracking system. He was able to compute estimates for the amount of glue-code that would be needed and how long that would take, but there were numerous risks that he wanted to take into account in his estimates. He didn’t know how to incorporate these risks, so he again turned to the CeBASE COTS Lessons Learned Repository for guidance. Despite searching for lessons on “planning”, “estimation”, and “risk management”, he could find nothing that would help him with his problem.*

*So he clicked on the “Ask An Expert” option and submitted a question on this subject. This began a dialogue with an expert in the area, who asked Joe questions about the specifics of his project and the risks he was concerned about. Finally, they came up with a plan that allowed Joe to quantify the*

*“cost” of a risk, as well as the “benefit” of reducing that risk. These figures could then be incorporated into the cost and benefit estimates for the task. Later, Joe was happy to find a new lesson in the repository that was synthesized from his discussion with the expert, so that others could benefit from what he had learned.*

The Ask An Expert feature is currently under implementation. It will offer an opportunity for peer-to-peer communication (left part of Figure 1). This dialogue is recorded and after the person who asked the question receives an answer, the log of this conversation is used to extract new lessons that will be added to the repository.

*Joe began using his new approach for quantifying risks on more projects, and over time built up some experience with it. However, he still had problems with using it in some situations. In particular, he often had trouble estimating the potential loss of having to entirely replace a component due to unforeseen problems with it. He wanted to bounce some ideas off someone, but nobody in his immediate work area was familiar enough with these issues to be able to give any useful feedback. So Joe went to the CeBASE site again and this time joined a discussion group on COTS estimation. He posted his questions about incorporating risk into estimation to the discussion group, and an interesting discussion ensued. Many people shared their experiences using different approaches to cost*

# Software Tech News Subscriber Survey

1. Which volume of the Software Tech News did you receive? STN 5:3 - Experience Based Mgmt.

2. When did you receive the newsletter? (*month/year*) \_\_\_\_\_

3. How satisfied were you with the **CONTENT** of the newsletter? (**Article Quality**)

☐ Very Satisfied    ☐ Satisfied    ☐ Neither Satisfied nor Dissatisfied    ☐ Dissatisfied    ☐ Very Dissatisfied

4. How satisfied were you with the **APPEARANCE** of the newsletter?

☐ Very Satisfied    ☐ Satisfied    ☐ Neither Satisfied nor Dissatisfied    ☐ Dissatisfied    ☐ Very Dissatisfied

5. How satisfied were you with the **OVERALL QUALITY** of the newsletter?

☐ Very Satisfied    ☐ Satisfied    ☐ Neither Satisfied nor Dissatisfied    ☐ Dissatisfied    ☐ Very Dissatisfied

6. How satisfied were you with the **ACCURACY** of the address on the newsletter?

☐ Very Satisfied    ☐ Satisfied    ☐ Neither Satisfied nor Dissatisfied    ☐ Dissatisfied    ☐ Very Dissatisfied

7. Approximately how much time and/or money did you save by using the product or service?

☐ Unable to estimate    \_\_\_\_\_ Estimated hours saved    \_\_\_\_\_ Estimated dollars saved

8. How did you request the product or service?

☐ Phone Call    ☐ E-mail    ☐ DACS Website    ☐ Subscription Form    Other \_\_\_\_\_

9. Would you order a **DACS** product or service again?

☐ Definitely    ☐ Probably    ☐ Not Sure    ☐ Probably Not    ☐ Definitely Not

## 10. Comments (Optional)

---

---

---

---

## Contact Information (Optional)

Name:	Position/Title:		
Organization:	Office Symbol:		
Address:			
City:	State:	Zip Code:	
Country:	E-mail:		
Telephone:	Fax:		
Organization Type:			
<input type="checkbox"/> Air Force	<input type="checkbox"/> Army	<input type="checkbox"/> Navy	<input type="checkbox"/> Other DoD _____
<input type="checkbox"/> Commercial	<input type="checkbox"/> Non-Profit	<input type="checkbox"/> Non-US	
<input type="checkbox"/> US Government	<input type="checkbox"/> FFR&D	<input type="checkbox"/> Other _____	

**The first 50 people to send in a completed survey will receive a FREE DoD/IT Acronym List CD.**

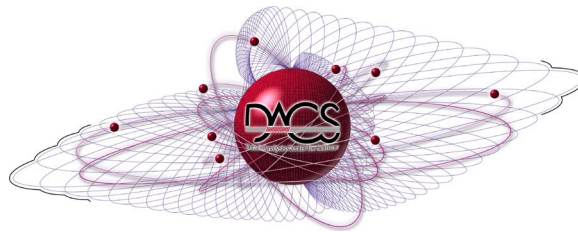
This valuable CD-ROM contains over 8,000 Department of Defense and Information Technology (DoD/IT) related acronyms. This Pentagon-shaped CD-ROM plays in your computer's regular CD drive.  
(Windows only)



▼ Fold Here ▼

---

## **Data & Analysis Center for Software (DACS)**



**<http://iac.dtic.mil/dacs/>**

▼ Fold Here ▼

---



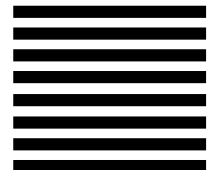
NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST-CLASS MAIL PERMIT NO. 120 ROME NY

POSTAGE WILL BE PAID BY ADDRESSEE

**DATA & ANALYSIS CENTER FOR SOFTWARE  
775 DAEDALIAN DR  
ROME NY 13449-0139**



*estimation on COTS projects, and some offered well-tested alternative approaches, such as the COTS Cost Estimation Model (COCOTS [2]). So Joe got some new ideas to try out on his projects, and everyone involved learned something. The lessons shared in the discussion that were best supported with real experience and data were captured as new lessons learned in the repository, so that others not involved in the discussion might learn from them too.*

Besides the peer-to-peer communication mechanism, we will offer other means for community support, such as group discussions and eWorkshops (moderated chat sessions where the log is recorded). These logs will be analyzed, mined, and new lessons will be derived and added to the repository, according to our approach of transforming “knowledge dust” into “knowledge

pearls” [5]. We will thus provide the process and the technology to support knowledge collection, organization, storage, evolution, and dissemination for the CBS community.

## 4. Conclusions

The COTS Lessons Learned repository aims at disseminating valuable knowledge and experience between practitioners involved in COTS-based development. Some lessons learned in this area have been published in papers, but have not been previously actively elicited and shared on a larger scale. By providing an online repository used by both experienced and less experienced people, it is possible to create a community of software engineers and managers that share this kind of knowledge and experience on a daily basis.

The COTS Lessons Learned Repository is waiting to serve your needs in CBS development. We appreciate your contributions with new lessons and any feedback you can give. Please visit us at <http://fc-md.umd.edu/ll/index.asp>.

## Acknowledgments

This work is partially sponsored by NSF grant CCR0086078 to the University of Maryland with subcontract to the Fraunhofer Center Maryland (FC-MD). Many thanks to Chris Abts from the University of Southern California (USC) for providing the material to initially populate the repository; to Forrest Shull and Patricia Costa from FC-MD and Dan Port from USC for their suggestions in developing the repository.

---

## References

1. Albert, C. and E. Morris, “Commercial Item Acquisition: Considerations and Lessons Learned”, <http://www.dsp.dla.mil/documents/cotsreport.pdf>
2. Abts, Chris, Boehm, B. and Bailey Clark, E., “COCOTS: a Software COTS-Based System (CBS) Cost Model - Evolving Towards Maintenance Phase Modeling,” *Proceedings of the Twelfth Software Control and Metrics Conference (ESCOM 2001)*, held at London, England, April 2001.
3. Basili, Victor, Boehm B., “COTS-Based Systems Top 10 List”, *IEEE Software*, May 2001, pp. 91-93.
4. Basili, Victor, R. Tesoriero, P. Costa, M. Lindvall, I. Rus, F. Shull, and M. Zelkowitz, “Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop”, *Proceedings of the 3rd International Conference on Product Focused Software Process Improvement, PROFES2001*, Kaiserslautern (Germany), September 2001.
5. Basili, Victor, P. Costa, M. Lindvall, M. Mendonca, C. Seaman, R. Tesoriero, and M. Zelkowitz, “An Experience Management System for a Software Engineering Research Organization”, in *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, December 2001.
6. Fox, Steve, M. Moore, “EOSDIS Core System (ECS) COTS Lessons Learned”, 25th Annual NASA Goddard Software Engineering Workshop, November 2000, [http://sel.gsfc.nasa.gov/website/sew/2000/SEW25\\_final\\_program.htm](http://sel.gsfc.nasa.gov/website/sew/2000/SEW25_final_program.htm)
7. Lewis, Patrick, Hyle P., Parrington M., Clark E., Boehm B., Abts C., Manners R., Brackett J., “Lessons Learned in Developing Commercial Off-the-Shelf (COTS) Intensive Software Systems”, FAA SERC Report, 2001.

**STN Vol. 5, No. 3**  
**In This Issue**

<b>The Role of Experience .....</b>	<b>3</b>
<b>Lessons-Learned Repository .....</b>	<b>4</b>
<b>Did We Lose Our Religion? .....</b>	<b>8</b>
<b>Habitats: Toward an Architect for Global Information Grid .....</b>	<b>12</b>
<b>A Stitch in Time .....</b>	<b>18</b>
<b>STN Subscriber Survey .....</b>	<b>22</b>

**Advertisement**

The *DoD Software Tech News* is now accepting advertisements for future newsletters. In addition to being seen by the thousands of people who subscribe to the *DoD Software Tech News* in paper copy the advertisement will also be placed on the Data & Analysis Center for Software's website (<http://iac.dtic.mil/dacs/>) exposing your product, organization, or service to hundreds of thousands of additional eyes.

Interested in learning more? For rates, layout information, and requirements contact:

Lon R. Dean, STN Editor  
Data & Analysis Center for Software  
P.O. Box 1400  
Rome, NY 13442-1400

Phone: 800-214-7921

Fax: 315-334-4964

E-mail: [news-editor@dacs.dtic.mil](mailto:news-editor@dacs.dtic.mil)

---

**Data & Analysis Center for Software**  
**P.O. Box 1400**  
**Rome, NY 13442-1400**

Return Service Requested

PRSRT STD  
U.S. Postage  
**PAID**  
Permit #566  
UTICA, NY